Article

Deep Learning-Based Bug Report Summarization Using Sentence Significance Factors

Youngji Koh [1], Sungwon Kang [1] and Seonah Lee [2,3,*]

1 School of Computing, Korea Advanced Institute of Science & Technology (KAIST), 291 Daehakro, Daejeon 34141, Korea; youngji@kaist.ac.kr (Y.K.); sungwon.kang@kaist.ac.kr (S.K.)
2 Department of Aerospace and Software Engineering, Gyeongsang National University, 201 Jinjudaero, Jinju-si 52828, Korea
3 Department of AI Convergence Engineering, Gyeongsang National University, 201 Jinjudaero, Jinju-si 52828, Korea
* Correspondence: saleese@gnu.ac.kr; Tel.: +82-55-772-1377

Abstract: During the maintenance phase of software development, bug reports provide important information for software developers. Developers share information, discuss bugs, and fix associated bugs through bug reports; however, bug reports often include complex and long discussions, and developers have difficulty obtaining the desired information. To address this issue, researchers proposed methods for summarizing bug reports; however, to select relevant sentences, existing methods rely solely on word frequencies or other factors that are dependent on the characteristics of a bug report, failing to produce high-quality summaries or resulting in limited applicability. In this paper, we propose a deep-learning-based bug report summarization method using sentence significance factors. When conducting experiments over a public dataset using believability, sentence-to-sentence cohesion, and topic association as sentence significance factors, the results show that our method outperforms the state-of-the-art method BugSum with respect to precision, recall, and F-score and that the application scope of the proposed method is wider than that of BugSum.

Keywords: bug report; bug tracking system; data-based software engineering; text summarization; open source software; sentence significance factor; software maintenance

check for updates

Citation: Koh, Y.; Kang, S.; Lee, S. Deep Learning-Based Bug Report Summarization Using Sentence Significance Factors. *Appl. Sci.* **2022**, *12*, 5854. https://doi.org/10.3390/app12125854

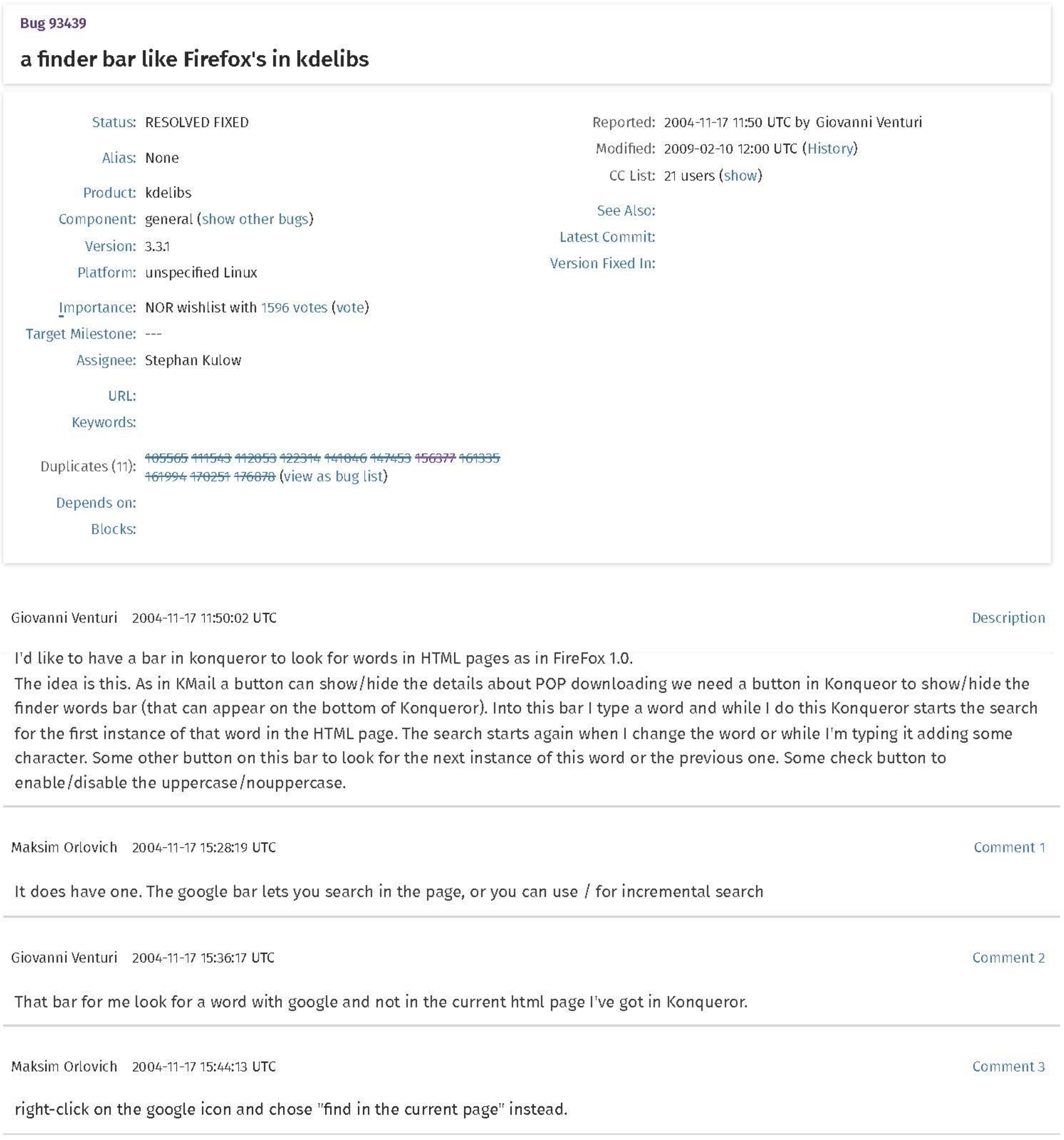
Academic Editor: Douglas O'Shaughnessy

Received: 16 March 2022
Accepted: 6 June 2022
Published: 8 June 2022

1. Introduction

Bug reports are an essential resource for long-term maintenance of software systems. Developers share information, discuss bugs, and fix associated bugs through bug reports [1]. Bug reports are managed by using bug tracking systems such as Trac and Bugzilla. Figure 1 shows an example of a bug report posted on the Bugzilla platform. As shown in Figure 1, the bug report contains a detailed description of the bug and a list of comments that contain the developers' discussion on how to reproduce or how to fix the bug. Bug reports are structured in the form of public posts with discussion sections similar to posts of social websites [2]. Because bug reports are written in the form of discussions among many people, developers often spend much time perusing lengthy bug reports and encounter difficulty obtaining the desired information.

To address these problems, various studies [3–10] have proposed methods for summarizing bug reports; however, most existing methods relied on either training data [3] or word frequencies [4], failing to produce high-quality summaries. As a method that utilizes information about the relationships among sentences in bug reports, Liu et al. [5] applied *sentence believability* to summarize bug reports. They considered that the discussions in bug reports are cross-validated among different developers who express their own opinions (i.e., evaluation behaviors) to evaluate others' comments. Comments in a bug report are either supported or disapproved by others. Liu et al. [5] showed that using believability scores can improve the quality of bug report summaries; however, for bug reports that do not contain such evaluated sentences, it is difficult to assign believability scores to sentences; therefore, the effect of their method is limited to the bug reports that contain evaluated sentences. To measure the importance of all sentences, Koh et al. [10] combined sentence believability score [5] with the TextRank-based text ranking score [11]; however, their approach is limited to the combination of two existing methods [5,11].


Bug 93439

a finder bar like Firefox's in kdelibs

Status:	RESOLVED FIXED	Reported:	2004-11-17 11:50 UTC by Giovanni Venturi
Alias:	None	Modified:	2009-02-10 12:00 UTC (History)
Product:	kdelibs	CC List:	21 users (show)
Component:	general (show other bugs)	See Also:	
Version:	3.3.1	Latest Commit:	
Platform:	unspecified Linux	Version Fixed In:	
Importance:	NOR wishlist with 1596 votes (vote)		
Target Milestone:	---		
Assignee:	Stephan Kulow		
URL:			
Keywords:			
Duplicates (11):	~~105565~~ ~~111543~~ ~~112053~~ ~~122314~~ ~~141046~~ ~~147453~~ ~~156377~~ ~~161335~~ ~~161994~~ ~~170251~~ ~~176878~~ (view as bug list)		
Depends on:			
Blocks:			

Giovanni Venturi 2004-11-17 11:50:02 UTC Description

I'd like to have a bar in konqueror to look for words in HTML pages as in FireFox 1.0.
The idea is this. As in KMail a button can show/hide the details about POP downloading we need a button in Konqueor to show/hide the finder words bar (that can appear on the bottom of Konqueror). Into this bar I type a word and while I do this Konqueror starts the search for the first instance of that word in the HTML page. The search starts again when I change the word or while I'm typing it adding some character. Some other button on this bar to look for the next instance of this word or the previous one. Some check button to enable/disable the uppercase/nouppercase.

Maksim Orlovich 2004-11-17 15:28:19 UTC Comment 1

It does have one. The google bar lets you search in the page, or you can use / for incremental search

Giovanni Venturi 2004-11-17 15:36:17 UTC Comment 2

That bar for me look for a word with google and not in the current html page I've got in Konqueror.

Maksim Orlovich 2004-11-17 15:44:13 UTC Comment 3

right-click on the google icon and chose "find in the current page" instead.

Figure 1. Example of a bug report.

In this paper, we propose a deep-learning-based bug report summarization method that utilizes the notion of sentence significance factors, where a *sentence significance factor* is a criterion for identifying important sentences. The main idea behind our proposed method is that users can obtain high-quality summaries without limiting the applicability by determining a suitable set of sentence significance factors and their weights for the target bug report.

We evaluate the proposed method with two datasets using three sentence significance factors: sentence-to-sentence cohesion, topic association, and believability. The experimental results show that our proposed method outperforms the state-of-the-art method of BugSum [5] with respect to precision, recall, and F-score and that the application scope of the proposed method is wider than that of BugSum [5].

The contributions of this paper are presented as follow: First, we proposed a comprehensive deep-learning-based bug report summarization method that encompasses diverse sentence significance factors. Second, we compared the proposed method with the state-of-the-art method BugSum with two benchmark datasets and showed that our comprehensive method that uses three sentence significance factors yields a better result than the result of BugSum and that our method works even when BugSum does not work.

This paper is organized as follows: Section 2 introduces existing studies related to bug report summarization. Section 3 describes our proposed method. Sections 4 and 5 evaluate the proposed method. Section 6 discusses threats to validity and implications of our study. Section 7 concludes our work and discusses future research challenges.

2. Related Work

Text summarization techniques can be classified into the abstractive approach and the extractive approach. Abstractive summarization generates a summary by using new phrases and sentences that may not appear in the original text. Extractive summarization organizes a summary by extracting a paragraph or sentence from the original text. Bug report summarization has been extensively studied in the direction of extractive summarization. The extractive summarization approach is highly relevant to this paper and is discussed in detail in the remainder of this section.

Rastkar et al. [6] proposed a bug report summarization method based on a supervised learning technique. They applied minutes and email tread summarization methods to bug reports, considering that bug reports are typically written in a dialog format. Rastkar et al. [6] prepared a handwritten summary of 36 bug reports for training and evaluation. Jiang et al. [7] confirmed that duplicated bug reports have textual similarity with original bug reports, and based on this relationship, proposed a bug report summarization method using the PageRank algorithm. They combined the method of Rastkar et al. [6] with the PageRank-based summarization method to improve the quality of bug report summaries. Additionally, they created new OSCAR data containing duplicated bug reports. Rastkar et al. [6] and Jiang et al. [7] trained the summarizer using the summaries that they created; however, in these studies, the quality of a summary is heavily dependent on training data, which requires annotators to exert massive manual efforts to create the training data.

Although many supervised-learning-based methods have been adopted, unsupervised-learning-based methods have been employed to avoid massive efforts for manually label data. Lotufo et al. [8] proposed a graph-based, unsupervised, summarization method that mimics human reading behaviors. They simulated human reading patterns to score sentences, connected the sentences according to similarities, and chose the sentences with the highest possibilities of being reached while randomly moving between two sentences. Li et al. [9] proposed an unsupervised, learning-based, summarization method using deep learning. They focused on predefined words and sentence types and scored sentences based on the weights of words; however, existing unsupervised approaches are sensitive to word frequency [4,9] and are likely to include redundant sentences that represent a similar topic in the summary, which would produce biased results.

Liu et al. [5] proposed a deep-learning-based, unsupervised, summarization method that measures the sentence believability score. They considered that bug reports contain evaluation behaviors, which are positive or negative opinions in the form of comments. They defined a sentence that evaluates a specific sentence as an *evaluator sentence* and a specific sentence that is evaluated by other sentences as an *evaluated sentence*. They assigned positive or negative weights to the *evaluated sentences* by other *evaluator sentences*. Based on the weights, they computed believability scores and improved summarization performance. Liu et al. [5] found that 28.5% of the 31,155 bug reports do not contain evaluation behaviors. In addition, in the case of bug reports with evaluation behaviors, the evaluated sentences comprised only 36.4% of the sentences on average. Thus, for bug reports that do not contain evaluation behaviors or bug reports that contain them only for a few sentences, their method will not work well.

Kiyoumarsi [12] summarized the document using many features at the sentence-based level, such as the sentence length, sentence position, sentence-to-sentence cohesion, and similarity to title. For example, similarity to title is calculated by comparing all the sentences of the document against the title of the document. They used a machine learning method based on these features to summarize documents; however, if a deep learning algorithm with sufficient relevant data were utilized, the summary quality would have been improved. Bug reports also contain sentence significance factors (e.g., sentence-to-sentence cohesion and similarity to title), which can be applied to increase the summary performance for bug reports that do not contain evaluation behaviors.

Koh et al. [10] proposed combining Liu et al.'s [5] sentence believability score with the TextRank-based text ranking score to measure the degree to which a sentence is important to generate high-quality summaries; however, they only focused on measuring sentence importance without considering how comprehensively the summary preserves the domain features of a bug report. In this paper, our method extracts domain features from a bug report and selects sentences that comprehensively preserve the domain features of a bug report. They also considered a few methods to measure sentence importance.

In this paper, we extend the existing methods [5,10] by considering sentence significance factors. The similarity between our study and the previous studies is that our study adopts the existing sentence significance factors (e.g., believability score [5]). The difference between our study and the previous studies is that, while one of the previous studies suggested a combination of two factors [10], our study devises an additional factor and suggests a combination of these three factors. In addition, our study systematically experiments with the three factors by changing their weights. In addition, our study reveals the impact of steps to summary quality by inserting and deleting steps in the experiment. Finally, our study analyzes the impact of the three factors to an application scope. Our study suggests a new direction to improve summary quality by combining various sentence significant factors of bug reports.

3. Proposed Method

An overview of our proposed method is shown in Figure 2. There are four inputs to our method. The first input is a bug report to be summarized, denoted as *BR*. The second input is a list of sentence significance factors, and the third input is a list of weights for the sentence significance factors. The fourth input is the length of the output summary. The output from our method is a summary of the bug report to be denoted as *Selected_Summary*.

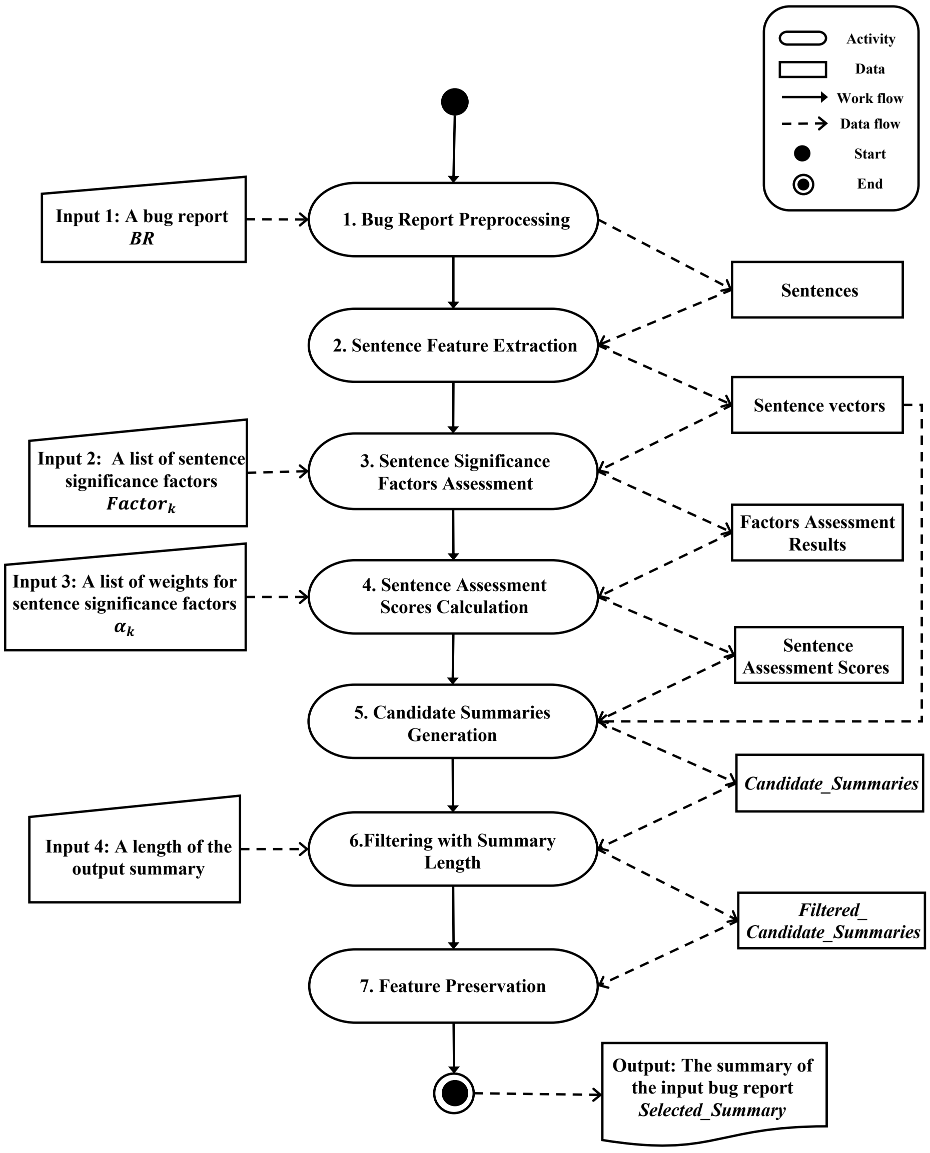

Activity
Data
Work flow
Data flow
Start
End
Input 1: A bug report *BR*
1. Bug Report Preprocessing
Sentences
2. Sentence Feature Extraction
Sentence vectors
Input 2: A list of sentence significance factors $Factor_k$
3. Sentence Significance Factors Assessment
Factors Assessment Results
Input 3: A list of weights for sentence significance factors α_k
4. Sentence Assessment Scores Calculation
Sentence Assessment Scores
5. Candidate Summaries Generation
Candidate_Summaries
Input 4: A length of the output summary
6.Filtering with Summary Length
Filtered_ Candidate_Summaries
7. Feature Preservation
Output: The summary of the input bug report *Selected_Summary*

Figure 2. Overview of the proposed method.

3.1. Sentence Significance Factors

The extractive-based, text summarization approach that selects sentences of high relevance or importance for summaries is based on employing a set of factors to generate coherent summaries that state the main idea of the given document [13]. Many factors have been proposed for automatic extractive text summarization by various researchers [5,12,13]. These factors include mean-TF-ISF, which is TF-IDF applied to a single document, sentence length, sentence position, similarity to title, similarity to keywords, sentence-to-sentence cohesion, etc. Since the quality of a generated summary is highly dependent on the selected factors, our method relies on the notion of sentence significance factors so that the user of the method can select the most relevant and important factors that determine the significance of sentences.

In our proposed method, the sentence significance factors are sentence believability, sentence-to-sentence cohesion, and topic association. For example, for the sentence "Firefox currently supports a wide number of stand alone e-mail applications (including its own Thunderbird client), when an e-mail link is clicked on in Firefox", the sentence believability score is zero if no sentence evaluates this sentence. The sentence-to-sentence cohesion score is 0.06, which is calculated using the text ranking algorithm. The topic association score is 0.88, which indicates the similarity between this sentence and the title.

3.2. Steps of the Proposed Method

Our proposed method consists of seven steps. In the first step, our method preprocesses a bug report and removes the noise contained in the bug report. In the second step, domain features are extracted from preprocessed sentences as vectors. In the third step, sentences are assessed with sentence significance factors. In the fourth step, sentence assessment scores are calculated. In the fifth step, candidate summaries are generated. In the sixth step, candidate summaries that are 30% of the length of the bug report are selected. In the seventh step, a summary that best preserves domain features from bug reports is selected.

The following subsections separately explain the steps of our proposed method.

3.2.1. Bug Report Preprocessing

Bug reports are real-world data that contain considerable noise [14]. Since data that contain noise do not produce accurate results, removing noise from data is necessary; therefore, first, code snippets and stack traces are removed. Second, bug reports are divided into sentences based on punctuation marks, and each sentence is tokenized following the software-specific regular expression [7]. Last, stop words are removed [15] and Porter stemming [16,17] is performed.

The output from this step is a list of preprocessed sentences. Assuming that the length of the list is n, we denote each sentence by sentence i, $1 \leq i \leq n$.

3.2.2. Sentence Feature Extraction

To extract domain features from sentences, an autoencoder network is employed. The basic principle of the autoencoder is illustrated in Figure 3. An auto-encoder consists of encoders and decoders. Each sentence is encoded to a latent vector, and then the latent vector is decoded to reconstruct the input sentence. The autoencoder is trained for the reconstructed sentence to be consistent with the input. The latent vector is considered a sentence vector that expresses the domain features of the sentence. The output from this step is a list of sentence vectors. The sentence vector of sentence i is denoted by S_i.

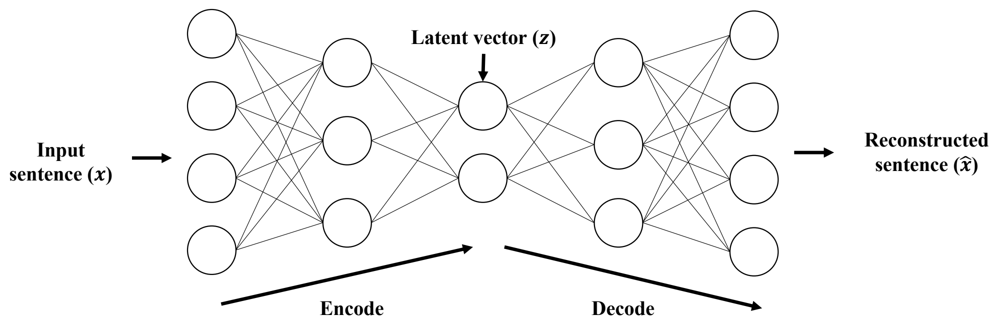


Figure 3. Structure of autoencoder.

3.2.3. Sentence Significance Factor Assessment

The second input to the proposed method is a list of sentence significance factors denoted by $< Factor_1, \ldots, Factor_m >$ when m is the number of sentence significance factors considered. To use these factors, metric M_k should be defined for each $Factor_k$, $1 \leq k \leq m$.

In Step 3, for each sentence vector S_i, each sentence significance factor $Factor_k$, $1 \leq k \leq m$, is evaluated using M_k. M_k ($Factor_k$ (S_i)) represents the score measured with M_k for Factor$_k$, $1 \leq k \leq m$, for the sentence vector S_i, $1 \leq i \leq n$. For example, we could define the metric M_k for $Factor_k$ as:

$$M_k(Factor_k(S_i)) = \begin{cases} 1 + \sum\limits_{j \in R(S_i)} f(S_i, S_j) & \text{if } |R(S_i)| > 0 \\ 1, & \text{if } |R(S_i)| = 0 \end{cases} \tag{1}$$

where $R(S_i)$ is the set of sentences in the input bug report that are related to sentence $R(S_i)$ in a certain way R and $f(S_i, S_j)$ is the value for the way S_i is related to sentence S_j.

3.2.4. Sentence Assessment Score Calculation

The third input is a list of weights for sentence significance factors denoted as $< \alpha_1, \ldots, \alpha_m >$ such that α_k is the weight for the sentence significance factor $Factor_k$.

The sentence assessment score of sentence vector S_i is denoted by $SAscore_i$ and is calculated as follows.

$$SAscore_i = \sum_{k=1}^{m} a_k \times M_k(Factor_k(S_i)) \tag{2}$$

where m is the number of input sentence significance factors, and α_k, $1 \leq k \leq m$, is a weight value between 0 and 1 such that the sum of α_k, $1 \leq k \leq m$, is 1. The closer α_k is to 1, the more important α_k is.

3.2.5. Candidate Summary Generation

For each bug report, Liu et al. [5] defined the notion of a full document vector to represent the domain features of the bug report *BR*. The full document vector is obtained by taking the sentence assessment scores as weights and summing the weighted average of the sentence vectors. The full document vector of bug report *BR* is denoted as *DF* and calculated as follows.

$$DF = \sum_{i=1}^{n} SAScore_i \times S_i \tag{3}$$

Therefore, *DF* represents the domain features of a bug report, as it combines the domain features of all the sentences with sentence assessment scores.

Candidate_Summaries is a set of candidate summaries, where a candidate summary, denoted as *CS*, is a set consisting of selected sentences from a bug report. *CS*, a candidate summary in *Candidate_Summaries*, which was obtained from Step 5, is applied to reconstruct the full document vector *DF*, denoted as $\widetilde{DF}$.

$$\widetilde{DF} = \sum_{i \in CS} SAScore_i \times S_i \tag{4}$$

3.2.6. Filtering with Summary Length

An input to this step is the length of the output summary, which is input 4 in our method. Previous studies have selected 25% to 30% of the length of a bug report for the best length of its summary [11] and widely utilized it in various existing bug report summarization methods [6,8]. In this paper, we use 30% of the length of a bug report as the most appropriate length of a summary. It means that our method select 30% of sentences in a bug report; however, our method will still work in the same way with a different value for the most appropriate length of a summary.

Therefore, in this step, first, a subset of *Candidate_Summaries*, to be denoted as *Filtered_Candidate_ Summaries*, is generated such that the subset contains the *CS*s, whose length is approximately 30% the length of the bug report.

3.2.7. Feature Preservation

In addition, a good summary should preserve the domain features of a full document as comprehensively as possible. To measure the degree to which a summary preserves domain features, we use reconstruction loss [5], denoted as δ, which is the number of features that exist in the bug report but do not exist in a summary; it is calculated as follows:

$$\delta = |MSE(DF, \widetilde{DF})| \tag{5}$$

In Equation (5), the mean squared error, denoted as MSE, between DF and $\widetilde{DF}$ is calculated using the following equation:

$$MSE = \frac{\sum_{i=1}^{d}(y_i - \widetilde{y}_i)^2}{d} \tag{6}$$

where d is the dimension of the feature vector and y_i and $\widetilde{y}_i$ are the vector components of DF and $\widetilde{DF}$, respectively.

The rationale for Equation (5) is that the closer $\widetilde{DF}$ is to DF, the more domain features of the bug report are contained in candidate summary CS, i.e., the closer δ is to 0, the more comprehensively CS summarizes the given bug report.

From the subset of *Candidate_Summaries* that contain the *CS*s whose lengths are approximately 30% the length of the bug report, the candidate summary with the minimum value of δ is selected as *Selected_Summary*.

4. Experimental Setup

Section 4.1 lists the research questions. Section 4.2 describes the experimental subjects. Section 4.3 explains the implementation of the proposed method for experiments. Section 4.4 selects sentence significance factors to be utilized for experiments and defines their metrics. Section 4.5 explains the designs of the experiments to answer the research questions. Section 4.6 defines the metrics to be used for evaluation of the proposed method.

4.1. Research Questions

To evaluate our method, we ask the following research questions:

RQ1. Does our proposed method of using sentence significance factors yield higher summary quality (i.e., F1-score) than that of BugSum?

There are two related research questions that we investigate to understand the aspects that affect summary quality:

RQ2. How do sentence significance factor assessments and feature preservation influence summary quality?

RQ3. How does summary quality change as the length of a summary changes?

As a consequence of its generality, our method has a wider applicability than BugSum, which relies on just believability. The following research question compares our method and BugSum in this respect.

RQ4. Is the application scope of our proposed method wider than that of the state-of-the-art method BugSum?

4.2. Experimental Subjects

We evaluate the proposed method over two popular benchmark datasets, i.e., Summary Dataset (SDS) [6] and Authorship Dataset (ADS) [18], consisting of 36 bug reports and 96 bug reports, respectively. Each bug report in the datasets is annotated by three annotators to ensure quality. The annotators were asked to summarize the report in approximately 25% of the length of the bug report using their own words. They were also asked to link each sentence in their summary to one or more sentences in the original bug report. For each bug report, the set of sentences linked by at least two annotators is referred to as the gold standard summary (GSS) [6].

4.3. Implementation of Our Method for Experiments

For the experiments, we implemented the proposed method using the existing tools for Steps 2, 3, and 7.

To implement Step 2 of our method, we employed an autoencoder network to extract domain features from preprocessed sentences as vectors. Liu et al. [5] used a bidirectional gated recurrent unit (Bi-GRU) as the encoder and decoder unit to capture the forward and backward context information of each sentence. Bi-GRU consists of a forward GRU and a backward GRU. They also train the autoencoder based on 31,155 bug reports. The proposed method uses trained autoencoders based on the method in [5].

To implement Step 3 of our method, we implemented the Sentence-to-Sentence Cohesion Assessment component and Topic Association Assessment component and applied the existing Sentence Believability Assessment component from [5].

To implement Step 6 of our method, we utilized the beam search algorithm to select the optimal CS, a candidate summary. It is extremely inefficient to evaluate all possible CS. We can significantly reduce the time and effort of evaluating all possible CS using the beam search algorithm while still ensuring a good quality summary and while maintaining the comprehensiveness of the bug report.

The beam search algorithm explores all candidate summaries by expanding the top-k promising choices, where k means beam size. Figure 4 shows how to generate a summary using beam search algorithms. For example, the bug report contains 4 sentences, and the dimension of the feature vector is 2. Each sentence vector can be denoted as S_1 = [0, 1], S_2 = [2, 0], S_3 = [0, 2], and S_4 = [3, 0]. Assuming that all of the sentence vectors have the same sentence assessment scores, which is 1, the full document vector is DF = [5, 3] based on Equation (3). When S_1 is in the CS, the reconstructed full document vector is calculated as $\widetilde{DF}$ = [0, 1] based on Equation (4). The reconstruction loss δ between two vectors is also calculated as 14.5 based on Equations (5) and (6). If the beam size is 2, only two nodes that have the lowest value of δ are expanded. Initially, the CS that contains S_2 or S_4 is selected. In the next step, similarly, the two candidate summaries, i.e., [S_1, S_4] and [S_3, S_4], are newly selected. The final CS with the lowest value of δ is selected as a summary and denoted as *Selected_Summary*.

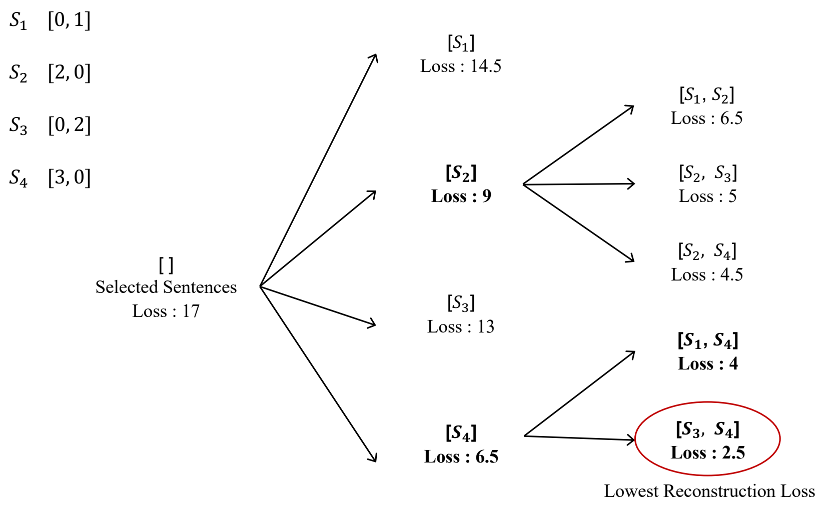


Figure 4. Process of the beam search algorithm.

4.4. Sentence Significance Factors Determination

In this experiment, three sentence significance factors are selected to evaluate the importance of a sentence: sentence believability, sentence-to-sentence cohesion, and topic association, which is similar to the title used in [12].

Considering that bug reports are mostly the form of a conversation, the factor based on sentence believability applied in [5] is promising; however, using sentence believability alone will have limitations in the scope of applicable bug reports because some bug reports have no sentences evaluated that are necessary to measure sentence believability. To augment the sentence believability factor, we select sentence-to-sentence cohesion and similarity to title, among the many candidates for sentence significance factors that we listed in Section 3.1, because they are commonly employed in general text summarization [12]. In addition, they reflect the following characteristics of a bug report: that everyone is free to comment, that the title summarizes the bug, and that irrelevant comments can be contained since many people freely write comments. In this case, the cohesion of a sentence is important for summarizing a bug report. Additionally, the title of a bug report implies the key point of the bug to its readers.

4.4.1. Sentence Believability Assessment

Sentence believability is computed based on evaluation behaviors. An evaluated sentence is assigned a believability score depending on the evaluator sentences. If an evaluator sentence gives a positive opinion about the evaluated sentence, this sentence is significant. The believability score of sentence i is denoted as $Bscore_i$.

$$Bscore_i = \begin{cases} 1 + \sum\limits_{j \in EAdj_i} Bscore_j \times OPscore_j & \text{if } |EAdj_i| > 0 \\ 1, & \text{if } |EAdj_i| = 0 \end{cases} \tag{7}$$

$Bscore_i$ is calculated as shown in Equation (7). A set of evaluator sentences that evaluate sentence i is denoted as $EAdj_i$. If sentence i is not evaluated by any other sentence, the believability score of this sentence is 1; however, if sentence j evaluates sentence i, the believability score is calculated using its evaluator sentences in $EAdj_i$.

The opinion score, denoted as $OPscore_j$, assesses the opinion of the evaluator sentence j toward the evaluated sentence i. An opinion score is measured using a support vector machine (SVM) classifier, which is a model pretrained with the dataset in [5]. The SVM classifier takes as input a sentence and predicts the probability that a negative opinion is expressed with a value between 0 and 1.

In this experiment, believability scores are calculated using the same dataset and the same SVM classifier that Liu et al. employed in [5] (https://github.com/HaoranLiu14/BugSum, accessed on 1 March 2022). Liu et al. trained the SVM classifier on a dataset containing 3000 sentences, which were collected from 31,155 bug reports and manually labeled.

4.4.2. Sentence-to-Sentence Cohesion Assessment

Sentence-to-sentence cohesion is computed using the TextRank algorithm [17], which assumes that a sentence similar to other sentences is a significant sentence.

To apply the TextRank algorithm, there is a need to build a graph using bug reports. The algorithm considers each sentence as a vertex and takes the similarity between two sentences as the weight of an edge. The values of all nodes are computed using the TextRank algorithm; however, the TextRank algorithm has limitations in that it cannot capture the entire meaning of a sentence because it computes a value only through the word frequencies.

Our method uses a TextRank algorithm based on sentence embedding [19] to capture the meaning of a sentence. First, the method computes the cosine similarity between two sentence vectors created in Step 2. The cosine similarity between two sentence vectors becomes the weight of an edge. Second, the TextRank algorithm is applied to calculate the value of each sentence. The TextRank value of sentence i is denoted as $Cscore_i$ and is calculated as shown in Equation (8)

$$Cscore_i = (1-d) + d \times \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)}} \times Cscore_j \tag{8}$$

For a given vertex V_i, $In(V_i)$ is the set of vertices that point to vertex V_i, and $Out(V_i)$ is the set of vertices to which vertex V_i points. The weight of the edge between two vertices V_i and V_j is denoted w_ij. A damping factor, which is denoted as d, is the probability of jumping from a given vertex to another random vertex in the graph. This factor is usually set to 0.85 [20]; we also use this value in our implementation.

The sentence is given an initial value of 1. The TextRank algorithm is run on the graph for several iterations until the TextRank value converges. The larger the number of sentences is with a similar meaning, the more important a sentence is, and the higher the TextRank value of the sentence.

4.4.3. Topic Association Assessment

Topic association is computed through the similarity of sentences to the title of a bug report. Titles typically reflect the key idea of a full document or at least contain words or phrases that are key to the topics; therefore, sentences closely related to the title are important enough to be included in the summary.

To evaluate the semantic closeness of each sentence with respect to its title, the similarity of the bug report's title and each sentence is computed by the cosine similarity measure [21], as shown in Equation (9). The cosine similarity of sentence i and the title is denoted as $Tscore_i$. The cosine similarity between the title and the sentence is calculated as follows, where A and B are the n-dimensional vectors representing a title and a sentence, respectively.

$$CosineSimilarity = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{9}$$

4.5. Experimental Design

4.5.1. Experiment for RQ1

To show the performance improvement of the proposed method, we compared our method with BugSum [5] on SDS and ADS.

4.5.2. Experiment for RQ2

The proposed method uses two steps that affect the accuracy of bug report summarization: sentence significance factor assessment in Step 3 and feature preservation in Step 7.

The sentence significance factors to be assessed in our method are sentence-to-sentence believability, sentence cohesion, and topic association. For feature preservation, we use the beam search algorithm, which will select the final output summary *Selected_Summary*. To analyze the effects of these two steps on the summary quality, we conducted two experiments.

In the first experiment, we analyzed the summary quality changes resulting from the selection of two steps. We intended to investigate whether each step actually contributes to summary quality improvement. If the step for feature preservation is excluded and the beam search algorithm does not apply, we selected the top 30% sentences with the highest sentence assessment scores.

In the second experiment, we identified the optimal weights to calculate the sentence assessment scores in Step 4 and analyze the summary quality changes based on the three sentence significant factors. Each factor is represented as a percentage. If the weight is 0, the corresponding assessment is skipped. When experimenting with only two of the three sentence significance factors, we determined the optimal weights by increasing the weights by 0.25. When experimenting with all three sentence significance factors, we determined the optimal weights by increasing the weights by 0.1.

4.5.3. Experiment for RQ3

An experiment was conducted to analyze the performance for various summary lengths. We generated a summary from 10% to 50% of the length of the bug report and analyzed the trend of summary quality as the summary length changes.

4.5.4. Experiment for RQ4

To evaluate whether our proposed method works even when BugSum [5] does not work because evaluated sentences do not exist in bug reports, we conducted an experiment as follows:

In the SDS, 13 of 36 bug reports do not have evaluated sentences. In the ADS, 50 of 96 bug reports do not have evaluated sentences. If there are evaluated sentences in a bug report, both BugSum [5] and our method can be applied; however, even if evaluated sentences do not exist in a bug report, our method still works, but BugSum will not work well. To confirm this point, we conduct an experiment with bug reports that do not have evaluated sentences. In the experiment, we compare summary quality using believability for Bugsum and the three sentence significance factors for our method, without using the beam search algorithm in both methods, to focus on the effects of the two methods where there are no evaluated sentences in the bug report.

4.6. Metrics

To evaluate the performance of bug report summarization methods, we used three metrics—precision, recall, and F1 score—which are commonly employed to measure the accuracy of bug report summaries. These metrics are calculated from the set of selected sentences denoted as Chosen and the set of the gold standard summary denoted as GSS. The metrics are expressed as follows:

$$\text{Precision} = \frac{|\text{Chosen} \cap \text{GSS}|}{|\text{Chosen}|} \tag{10}$$

$$\text{Recall} = \frac{|\text{Chosen} \cap \text{GSS}|}{|\text{GSS}|} \tag{11}$$

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

Especially, for RQ1, we measure the performance of bur report summarization by using ROUGE-1. ROUSE-1 calculates how many words are overlapped between Chosen and GSS. The metric is expressed as follows, where x and y represent words.

$$\text{ROUGE-1} = \frac{|\{x|x \in \text{Chosen} \cap x \in \text{GSS}\}|}{|\{y|y \in \text{GSS}\}|} \tag{13}$$

5. Experimental Results

In this section, we analyze the evaluation results of the proposed method for each research question.

5.1. RQ1

Table 1 shows the precision, recall, and F1-score when the bug reports in the SDS are summarized, and Table 2 shows the precision, recall, and F1-score when the bug reports in the ADS are summarized.

Table 1. Quality comparison of summarizing bug reports in the SDS.

	Precision	Recall	F1 Score
BugSum	0.501	0.429	0.442
Proposed Method	0.525	0.449	0.463

Table 2. Quality comparison of summarizing bug reports in the ADS.

	Precision	Recall	F1 Score
BugSum	0.506	0.507	0.475
Proposed Method	0.534	0.535	0.502

Our method outperforms BugSum [5] by approximately 2% in terms of the precision, recall, and F1 score on the SDS. Our method also outperforms the ADS by approximately

3%. When we calculate *p*-value from a *t*-test on the SDS, the proposed method is not statistically significant compared to BugSum ($p > 0.05$). When we calculate *p*-value from a *t*-test on the ADS, the proposed method is statistically significant compared to BugSum ($p < 0.05$). This means that, if we consider two datasets together, our proposed method will show its statistical significance, because of the increased size of samples; therefore, we interpret these results as indicating that additional factors, sentence-to-sentence cohesion, and topic association are useful.

Since the scores in the tables are around 0.5, one might think that a purely random coin-flipping approach would yield similar results; however, we remind that approximately 25% of the sentences in a bug report are included in GSS (see Section 4.2) and 30% of the sentences are selected automatically by both BugSum and our proposed method (see Section 3.2.6). We simulated the random cases. Our simulator randomly selected 25 out of 100 numbers as GSS, and randomly selected 30 out of 100 numbers again. We calculated precision, recall, and F1 scores from the result. We repeated the simulation 1000 times and averaged precision, recall, and F1 scores. The averaged precision score is 0.245, the averaged recall score is 0.294, and the averaged F1 score is 0.267. Both BugSum and our proposed method yielded much higher scores than the random selection.

Furthermore, we measured the performance improvement of the proposed method by using ROUGE-1. Table 3 shows the ROUGE-1 scores of BugSum and our method.

Table 3. Quality comparison of summarizing bug reports in terms of ROUGE-1.

	ROUGE-1 for SDS	**ROUGE-1 for ADS**
BugSum	0.44	0.31
Proposed Method	0.47	0.48

Our method yields 3% higher ROUGE-1 score than that of BugSum for the SDS and 17% higher score for the ADS, which clearly shows that our method outperforms BugSum in terms of the ROUGE-1 score.

RQ1. *Our proposed method of using sentence significance factors yields higher F1-score than that of BugSum by 2.1% on the SDS and 2.7% on the ADS. Our method also yields higher ROUGE-1 score than that of BugSum by 3% on the SDS and 17% on the ADS.*

5.2. RQ2

Table 4 shows the summary quality changes with the SDS and ADS by including and excluding Steps 3 and 7, as discussed in Section 4.5.2. In Table 4, the sign "×" indicates that the relevant step is excluded in the method applied, and the sign "✓" indicates that the relevant step is included in the method applied. The experimental results show that the F1 scores are the highest when the proposed method includes Steps 3 and 7. These steps are useful for summarizing bug reports.

Table 4. Comparison of the effects of steps 3 and 7.

Datasets	**Sentence Significance Factors Assessment**	**Feature Preservation (Beam Search Algorithm)**	**Precision**	**Recall**	**F1 Score**
SDS	×	✓	0.481	0.414	0.426
	✓	×	0.503	0.458	0.461
	✓	✓	0.525	0.449	**0.463**
ADS	×	✓	0.487	0.486	0.456
	✓	×	0.468	0.527	0.473
	✓	✓	0.534	0.535	**0.502**

We discover that excluding the step for feature preservation causes a decrease in the F1 score by 2.9% with the ADS dataset. In contrast, the F1 score decreases by 0.2% on SDS. It means the step has less impact on the SDS. The reason is that the average number of sentences per bug report in the ADS is only 39, while the average number of sentences is 65 in the SDS. In these long bug reports, there is sufficient information to generate a summary, e.g., evaluation behaviors occur more frequently in long bug reports; therefore, bug reports in SDS are less affected by this step; however, sentence significance factor assessment has an impact on both the SDS and ADS. For example, the F1 score increases by 3.7% on the SDS, and the impact on the ADS is 4.6%.

The results show that sentence significance factor assessment and feature preservation have a positive influence regardless of the dataset, but sentence significance factor assessment has a greater impact on summary quality than feature preservation.

The second experimental results are shown in Tables 5 and 6. Table 5 shows the impact of the sentence significance factors on the summary quality on the SDS. The experimental results show that the summary quality is the lowest when only sentence believability is applied and is the highest when all three sentence significance factors are applied. The number of applicable bug reports decreased when only sentence believability is considered, while more bug reports are subject to summarization when other sentence significance factors are combined.

Table 5. Impact of sentence significance factors on summary quality on SDS.

	Precision	Recall	F1 Score
Bscore ($\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 0$)	0.501	0.429	0.442
Cscore ($\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0$)	0.508	0.436	0.449
Tscore ($\alpha_1 = 0, \alpha_2 = 0, \alpha_3 = 1$)	0.520	0.444	0.458
Bscore + *Cscore* ($\alpha_1 = 0.75, \alpha_2 = 0.25, \alpha_3 = 0$)	0.516	0.441	0.455
Bscore + *Tscore* ($\alpha_1 = 0.5, \alpha_2 = 0, \alpha_3 = 0.5$)	0.524	0.447	0.462
Cscore + *Tscore* ($\alpha_1 = 0, \alpha_2 = 0.5, \alpha_3 = 0.5$)	0.517	0.442	0.456
Bscore + *Cscore* + *Tscore* ($\alpha_1 = 0.1, \alpha_2 = 0.1, \alpha_3 = 0.8$)	0.525	0.449	**0.463**

Table 6. Impact of sentence significance factors on summary quality on ADS.

	Precision	Recall	F1 Score
Bscore ($\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 0$)	0.506	0.507	0.475
Cscore ($\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0$)	0.503	0.508	0.474
Tscore ($\alpha_1 = 0, \alpha_2 = 0, \alpha_3 = 1$)	0.522	0.511	0.484
Bscore + *Cscore* ($\alpha_1 = 0.75, \alpha_2 = 0.25, \alpha_3 = 0$)	0.523	0.526	0.494
Bscore + *Tscore* ($\alpha_1 = 0.75, \alpha_2 = 0, \alpha_3 = 0.25$)	0.530	0.531	0.498
Cscore + *Tscore* ($\alpha_1 = 0, \alpha_2 = 0.75, \alpha_3 = 0.25$)	0.522	0.521	0.490
Bscore + *Cscore* + *Tscore* ($\alpha_1 = 0.8, \alpha_2 = 0.1, \alpha_3 = 0.1$)	0.534	0.535	**0.502**

Table 6 shows the impact of sentence significance factors on the summary quality on the ADS. The lowest precision and F1 scores are obtained when only sentence-to-sentence cohesion is utilized, and recall is the lowest when only sentence believability is employed. The summary quality is the highest when all three sentence significance factors are utilized, similar to the case of SDS.

RQ2. *The steps of sentence significance factor assessment and feature preservation have a positive influence on summary quality (i.e., F1-score). In the step of sentence significance factor assessment, the summary quality (i.e., F1-score) is the highest when all three sentence significance factors are combined.*

5.3. RQ3

Figure 5 shows the experimental results with the SDS. Figure 6 shows the experimental results with the ADS. The x-axis represents the length of the summary, and the y-axis shows the values of three metrics: precision, recall, and F1 score.

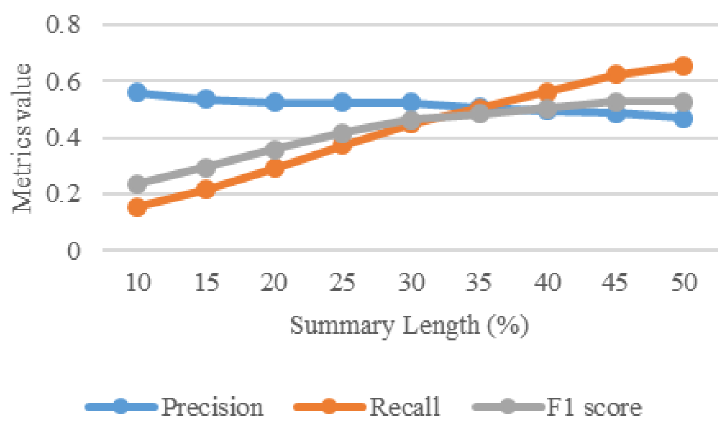


Figure 5. Summary quality changes with varied summary lengths: SDS dataset.

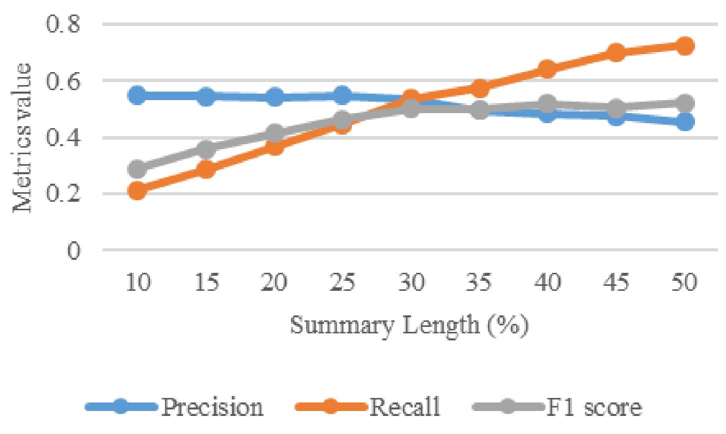


Figure 6. Summary quality changes with varied summary lengths: ADS dataset.

As a result of the experiment, the precision decreases by approximately 10% as the length of the summary increases; however, the recall increases to 50%. When the length of the summary is half that of the bug report, the recall reaches 0.656 and 0.726 with the SDS and ADS, respectively. The F1 score also increases to 0.504 and 0.519 with the SDS and ADS, respectively, when the length of the summary statement was 40% of the bug report. The increase in F1 score is less than 3%, even if the length of the summary further increases. The proposed method covers more than half of the sentences in the ground truth summary with only 40% of the length. The proposed method works well with the varied summary length.

RQ3. *Summary quality does not change much when the length of a summary is more than 30% of the length of a report. However, summary quality increases until the length of summary becomes 40% of the length of a bug report in both the SDS and the ADS.*

5.4. RQ4

In SDS and ADS, 64% and 48% of all bug reports contain evaluated sentences, respectively. When bug reports contain evaluated sentences, sentence-to-sentence cohesion and topic association are not very useful in improving quality of the summary. In the case, quality of summary can be improved to some extent only by considering sentence believability alone. It means that our proposed method could work better than BugSum, especially when bug reports do not have evaluated sentences.

Tables 7 and 8 show the evaluation results with the bug reports in the SDS and ADS, respectively, that do not have evaluated sentences. Experimental results show that BugSum produces low-quality summaries when there are no evaluated sentences. If evaluated sentences do not exist, all sentences in the bug report have the same believability score of 1; therefore, BugSum selects sentences until the selected sentences comprise 30% of the length of the bug report.

We performed random selection in these experiments and obtained the numbers for BugSum in Tables 7 and 8; however, our proposed method is not affected by the lack of evaluated sentences and shows higher F1 scores than those of BugSum by 12.4% and 14.2% with the SDS and ADS, respectively. When we calculate *p*-value from a *t*-test on the SDS, the proposed method is statistically significant compared to BugSum ($p < 0.05$). The *p*-value from a *t*-test on the ADS is also less than 0.05.

Table 7. Quality comparison of summarizing bug reports with no evaluated sentences in the SDS.

	Precision	**Recall**	**F1 Score**
BugSum	0.386	0.298	0.331
Proposed Method	0.542	0.339	0.455

Table 8. Quality comparison of summarizing bug reports with no evaluated sentences in the ADS.

	Precision	**Recall**	**F1 Score**
BugSum	0.394	0.397	0.373
Proposed Method	0.532	0.555	0.515

RQ4. *When there are no evaluated sentences in bug reports, our proposed method of using sentence significance factors yields a higher F1-score than that of BugSum by 12.4% on the SDS and 14.2% on the ADS.*

6. Discussion

For our discussion, we investigate one sample and show the relevant confusion matrix to explain the performance of our proposed method. We also discuss threats to validity and the implications of our study.

6.1. Confusion Matrix of One Summary Case

Table 9 shows the confusion matrix for the 9th bug report summary results in the SDS dataset. The bug report consists of a total of 17 sentences. Two columns "Actually positive" and "Actually negative" indicate whether or not each sentence is selected in a golden summary, and two rows "Predicted positive" and "Predicted negative" indicate whether or not each sentence is selected by the proposed method as a summary sentence. As shown in Table 9, when the proposed method is applied, the precision value (TP/TP + FP) is generally higher than that of recall (TP/TP + FN). It is because of the percentage of the sentences that are selected as summary sentences.

Table 9. Confusion matrix of 9th bug report of SDS.

	Actually Positive (8)	**Actually Negative (9)**
Predicted Positive (5)	True Positive (3)	False Positive (2)
Predicted Negative (12)	False Negative (8)	True Negative (4)

In the case of the summary sentences of SDS and ADS data, three people selected the sentences that contain key information in a bug report and then two or more people had to agree to select sentences as summary sentences; therefore, the percentage of the summary sentences in a golden summary does not account for 30% of the length of a bug report. In contrast, the proposed method always selects 30% of the sentences in a bug report as summary sentences. Due to this difference, the precision is always higher than recall; therefore, to evaluate the proposed method more accurately, we need to build a more elaborated experimental data.

6.2. Threats to Validity

In this section, we identify potential threats to the validity of our study.

- Internal validity: The performance difference between our proposed method and BugSum is not huge; however, the experimental result for RQ2 shows that the difference comes from the combination of three sentence significance factors. In addition, the experimental result for RQ4 shows a significant difference between performance of our proposed method and that of BugSum, based on the bug reports without evaluated sentences.
- External validity: In our evaluation, we only used two datasets, SDS and ADS. By using the datasets previously used in other summary methods, we could objectively evaluate the proposed method; however, we need to evaluate our proposed method with larger datasets. It is challenging to create large datasets, however, because humans have to create the golden summaries.
- Construct validity: We regarded the summary quality to be represented by the value of F1-score. To compensate for this threat, we used a different metric (e.g., ROUGE-1 score) to evaluate summary quality for RQ1.
- Conclusion validity: Although the performance difference between our proposed method and BugSum in the result of RQ1 is not huge, we showed that the difference is not a coincidence by showing the result of combining several sentence significance factors in RQ2. We also analyzed the influence of each step in RQ2.

6.3. Implication

This paper suggests a new direction that considering a combination of various sentence significance factors will help summarize a bug report more effectively than relying on a single sentence significance factor. This paper also shows a promising experimental result in this direction. We expect this study to be a stepping stone to combine several sentence significance factors for improving summary quality.

Based on our research results, practitioners, researchers, and tool builders can take the following further actions. First, practitioners can improve the summary quality of a bug report by referring to the steps in Figure 2 as a framework of utilizing various factors for improving summary quality. They need to search optimal weights of several sentence significance factors and combine them according to Equation (2) in this paper. Second, researchers can conduct further research to find, combine, and test more sentence significance factors, because we have limited our evaluation to three sentence significance factors. Furthermore, since it is already common to adopt an ensemble method for improving classification accuracy and some adopt an ensemble method for summarization, researchers can extend this concept and propose an ensemble method for bug summarization. Finally, tool builders can adopt our method to develop tools for bug summarization. Especially, they can consider combining several sentence significance factors or combining several summary techniques.

7. Conclusions

To improve the quality of bug report summaries and extend the scope of bug reports that can be summarized, this paper proposed a deep-learning-based bug report summarization based on the notion of sentence significance factors. A comparison of the state-of-the-art bug report summarization method BugSum with the popular benchmark datasets, SDS and ADS, and three sentence significance factors, i.e., sentence believability, sentence-to-sentence cohesion, and topic association, which showed that the proposed method yields 3% higher summary quality in terms of precision, recall, and F1 score and approximately 12∼14% higher summary quality when there are no evaluated sentences in bug reports.

The three sentence significance factors utilized in the evaluation of the proposed method were selected without considering the characteristics of the summarized bug reports. If the characteristics of bug reports are considered when determining a set of sentence significance factors, we believe that the summary quality will greatly improve. As further research, we will explore the corelationships between the characteristics of bug reports and sentence significance factors. We also plan to examine how to determine the optimal weights of sentence significance factors to calculate sentence assessment scores, and for more accurate experiments, we will build a dataset containing more than 10,000 bug reports. For the experiments, we utilized an autoencoder to extract the domain features of sentences since the goal of this paper is not to improve the deep learning algorithm but to improve the sentence assessment. For more sophisticated summarization, we will also use bidirected encoder references from transformers (BERT) [22]. In addition, we will consider performing a qualitative analysis in our future studies.

Author Contributions: Conceptualization, Y.K. and S.L.; methodology, Y.K.; software, Y.K.; validation, Y.K., S.K. and S.L.; formal analysis, Y.K. and S.K.; investigation, S.L.; resources, Y.K.; data curation, Y.K.; writing—original draft preparation, Y.K. and S.K.; writing—review and editing, S.L.; visualization, Y.K.; supervision, S.K.; project administration, S.K. and S.L.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (NRF-2021R1A2C1094167). This research was also supported by the "Regional Innovation Strategy (RIS)" through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (MOE) (2021RIS-003).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Our replication package is publicly available at https://github.com/youngji-koh/Bug-Report-Summarization-using-SSF (accessed on 9 May 2022).

Acknowledgments: Samal Mukhtar conducted the experiment to measure the performance of BugSum and our proposed method in terms of ROUGE-1.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDS	Summary Dataset
ADS	Authorship Dataset

References

1. Boehm, B.; Rombach, H.D.; Zelkowitz, M.V. *Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005.
2. Kim, W.; Jeong, O.R.; Lee, S.W. On social Web sites. *Inf. Syst.* **2010**, *35*, 215–236. [CrossRef]
3. Rastkar, S.; Murphy, G.C.; Murray, G. Summarizing software artifacts: A case study of bug reports. In Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering, Cape Town, South Africa, 2–8 May 2010; Volume 1, pp. 505–514.
4. Radev, D.R.; Jing, H.; Styś, M.; Tam, D. Centroid-based summarization of multiple documents. *Inf. Process. Manag.* **2004**, *40*, 919–938. [CrossRef]

5. Liu, H.; Yu, Y.; Li, S.; Guo, Y.; Wang, D.; Mao, X. Bugsum: Deep context understanding for bug report summarization. In Proceedings of the 28th International Conference on Program Comprehension, Seoul, Korea, 13–15 July 2020; pp. 94–105.
6. Rastkar, S.; Murphy, G.C.; Murray, G. Automatic summarization of bug reports. *IEEE Trans. Softw. Eng.* **2014**, *40*, 366–380. [CrossRef]
7. Jiang, H.; Nazar, N.; Zhang, J.; Zhang, T.; Ren, Z. Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates. *Int. J. Softw. Eng. Knowl. Eng.* **2017**, *27*, 869–896. [CrossRef]
8. Lotufo, R.; Malik, Z.; Czarnecki, K. Modelling the 'hurried' bug report reading process to summarize bug reports. *Empir. Softw. Eng.* **2015**, *20*, 516–548. [CrossRef]
9. Li, X.; Jiang, H.; Liu, D.; Ren, Z.; Li, G. Unsupervised deep bug report summarization. In Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), Gothenburg, Sweden, 27 May–3 June 2018.
10. Koh, Y.; Kang, S.; Lee, S. Bug Report Summarization using Believability Score and Text Ranking. In Proceedings of the 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Korea, 13–16 April 2021; pp. 117–120.
11. Kim, B.; Kang, S.; Lee, S. A Weighted PageRank-Based Bug Report Summarization Method Using Bug Report Relationships. *Appl. Sci.* **2019**, *9*, 5427. [CrossRef]
12. Kiyoumarsi, F. Evaluation of automatic text summarizations based on human summaries. *Procedia-Soc. Behav. Sci.* **2015**, *192*, 83–91. [CrossRef]
13. Qaroush, A.; Farha, I.A.; Ghanem, W.; Washaha, M.; Maali, E. An efficient single document Arabic text summarization using a combination of statistical and semantic features. *J. King Saud Univ.-Comput. Inf. Sci.* **2019**, *33*, 677–692. [CrossRef]
14. Xuan, J.; Jiang, H.; Hu, Y.; Ren, Z.; Zou, W.; Luo, Z.; Wu, X. Towards effective bug triage with software data reduction techniques. *IEEE Trans. Knowl. Data Eng.* **2014**, *27*, 264–280. [CrossRef]
15. Doyle, D. Default English Stopwords List. 2017. Available online: https://www.ranks.nl/stopwords (accessed on 1 March 2022).
16. Porter, M.F. An algorithm for suffix stripping. *Program Electron. Libr. Inf. Syst.* **1980**, *14*, 130–137. [CrossRef]
17. Mihalcea, R.; Tarau, P. Textrank: Bringing order into text. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, Barcelona, Spain, 25–26 July 2004; pp. 404–411.
18. Jiang, H.; Zhang, J.; Ma, H.; Nazar, N.; Ren, Z. Mining authorship characteristics in bug repositories. *Sci. China Inf. Sci.* **2017**, *60*, 1–16. [CrossRef]
19. Jeong, S.; Kim, J.; Kim, H. Document Summarization Using TextRank Based on Sentence Embedding. *J. KIISE* **2019**, *3*, 285–289. [CrossRef]
20. Page, L.; Brin, S.; Motwani, R.; Winograd, T. *The PageRank Citation Ranking: Bringing Order to the Web*; Technical Report; Stanford InfoLab: Stanford, CA, USA, 1999.
21. Jones, K.S. Automatic summarizing: Factors and directions. In *Advances in Automatic Text Summarization*; MIT Press: Cambridge, MA, USA, 1999; pp. 1–12.
22. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.